Projet Microcontrôleurs

Adrián BULNES, David FUENMAYOR

May 13, 2011

1 Introduction

Le document illustre d'une manière succinte les processus d'analyse, concéption et dévéloppement d'un système de commande pour un moteur à courant continue. Deux differents approches sont expliqués pour la mise en oeuvre de la commande numérique dans le microcontrôleur, sujet aux contraintes imposées par la construction de la maquette dans laquelle le microcontrôleur est embarqué.

2 Commande numérique PID

Il s'agit d'un mécanisme générique d'asservissement amplement utilisé dans l'industrie, et qui calcule la valeur de l'erreur comme la différence entre une variable du processus à réguler et une certaine signal de consigne fourni au préalable. L'algorithm de calcul inclut trois paramètres constantes: P, I et D, qui pondèrent les correspondantes parties proportionnel, intégrale et dérivatif de la signal d'erreur:

$U(z) = K_p.E(z)+K_I.\frac{z}{z-1} \cdot E(z)+K_D.\frac{z-1}{z} \cdot E(z)$

Vu que cette configuration du controleur n'est pas forcément la plus approprié dû aux problèmes entraînés par la dérivation du signal d'erreur, on a choisi la méthode de réglage de Takahashi

Après le travail d'identification du système, on a pu obtenir les paramètres des régulateurs P et PI, on ne fera pas le point sur le régulateur PID, vu que ses résultats expérimentales n'ont pas été assez satisfaisants.

$a = \frac{K}{Tau}$: Gain statique/Constante du temps du système

T_R : Retard Pur

2.1 Approche I

Pour le calcul du gain statique on considère les deux différents approches. Dans le prémier, on utilise le signal du encoder comme source d'horloge pour le microcontrôleur, on a vu qu'à la vitesse maximale du moteur, on obtient 8 flancs provenant du capteur (encoder) pour chaque période d'echantillonage, qui équivalent à une valeur de 255 pour le PWM, donc on utilisera un gain pour le capteur de $K_{capteur} = \frac{255}{8} \approx 32$ De cette façon, en multipliant le nombre de flancs de l'encoder par $K_{capteur}$ pour établir la sortie du système, on arrive a obtenir un gain statique unitaire $K = 1$

La période d'echantillonage utilisée corresponde à la constante de temps mesuré du système: $Tau = 32ms$

D'où les équations pour chacun des trois contrôleurs après quelques calculations:

2.1.1 Contrôleur Type P:

- $K_p = 0,88$
- $u_k = 0,88*(c_k - y_k)$

Régulateur	Cofficients de réglage	
	Réponse indicielle: a, T_R $T_R \geq 0.5\,Te$	Phénomène de pompage k_0, T_0
$u_k = K_p(c_k - y_k)$	$K_p = \frac{1}{a(T_R + T_e)}$	$K_p = 0.5K_0$
$u_k = u_{k-1}$ * $+K_p(y_{k-1} - y_k)$ $+K_I T_e(c_k - y_k)$ * l'action proportionnelle n'agit pas sur la consigne	$K_p = \frac{0.9}{a(T_R + 0.5T_e)} - 0.5K_I T_e$ $K_I = \frac{0.27}{a(T_R + 0.5T_e)^2}$	$K_p = 0.45K_0 - 0.5K_I T_e$ $K_I = 0.54\frac{k_0}{T_0}$
$u_k = u_{k-1}$ ** $+K_p(y_{k-1} - y_k)$ $+K_I T_e(c_k - y_k)$ $+\frac{K_D}{T_e}(2y_{k-1} - y_k - y_{k-2})$ ** les actions proportionnelle et dérivée n'agissent pas sur la consigne	$K_p = \frac{1.2}{a(T_R + T_e)} - 0.5K_I T_e$ $K_I = \frac{0.6}{a(T_R + 0.5T_e)^2}$ $K_D = \frac{0.5}{a}$ ou $\frac{0.6}{a}$ si $\frac{T_R}{T_e}$ entier	$K_p = 0.6K_0 - 0.5K_I T_e$ $K_I = 1.2\frac{k_0}{T_0}$ $K_D = \frac{3}{40}k_0 T_0$

Figure 1: Réglage Takahashi

2.1.2 Contrôleur Type PI:

- $K_p = 0,83$
- $K_I = 7,5$
- $u_k = u_{k-1} + 0,83*(y_{k-1} - y_k) + 0,03 * (c_k - y_k)$

Afin d'optimiser la performance du microcontrôleur, on evite les opérations aritmétiques avec des nombres de virgule flotant, de façon que pour le régulateur type P, on fait l'approximation: $0,88 \approx 0,875 = 0,5 + 0,25 + 0,125$ donc on a:

$0,88 * nombre \approx nombre >> 1 + nombre >> 2 + nombre >> 3$

2.2 Aproche II

Dans cet approche, on utilise le TIMER 1 pour mesurer le temps qui passe entre deux flancs montants consécutifs du signal de l'encoder. Pour faire cela, il est necessaire de générer une interruptions pour chaque flanc. Étant donné la restriction sur l'utilisation du port B du microcontrôleur (affichage LCD) on a dû improviser un système de détection par software de flancs montants sur le port RD1. Le grand avantage cette fois-ci, c'est d'avoir une plus grande résolution avec le capteur (jusqu'a 255 flancs au lieu de 8 pour T_e)

Aussi, avec cet approche, il est possible d'utiliser une période d'echantillonnage T_e beaucoup plus petite, ce qui permet d'avoir des meilleures performances pour l'asservisement.

References

[Polycopies cours d'automatique linéaire echantilloné] ENSMM

[http://www.microchip.com] Documentation Microchip

3 Code implanté sur le microcontrôleur

Le code utilisé pour effectuer la régulation pour les deux approches fut:

Algorithm 1 Approche-I

```
/*******************************************************/
// µ-projet de commande par microcontrôleur
// ENS2M avril 2011
// Adrian BULNES
// David FUENMAYOR
/*******************************************************/

    #include <pic.h>
    #include "afficheur.h"
    #define SwAffichConsigne RE3
    #define SwAffichErreur RD4
    #define SwAffichCommande RD6
    #define SwAffichRetour RD7
    #define Capteur RD1
    #define Appuye 0
    #define MoteurON RD5
    #define Kp 1
    static char consigne;
    static char erreur;
    static char commande;
    static char retour;
    static char encoder;
    static char vitesse;
    //Timer 0 - interrupt 5ms pour faire commande
    void setupT0() {
        //Fosc=4MHz, Fcy=1MHz, Tcy=1us
        //interrupts chaque 128*256us = 32ms@ 128prescaler
        T0CS=0; //Horloge Interne
        PSA=0;
        PS2=1;
        PS1=1; //Prescaler 128
        PS0=0;
        TMR0=0; //Initialization a zero
        T0IF=0; //Interrupt flag to zero
        T0IE=1; //Enable interrupt
}
    //Timer1 - compteur flancs encoder sans prescaler en RC5/T1CKI
    void setupT1() {
        T1CKPS1=0; //Pas de prescaler
        T1CKPS0=0; //Pas de prescaler
        TMR1CS=1; //Mode compteur
        T1SYNC=1; //Synchronisation habilété
        TMR1H=0; //Initialization a zero
        TMR1L=0; //Initialization a zero
        TMR1IF=0;
        PEIE=1; //Enable peripheral interrupt
        TMR1IE=1;
        TMR1ON=1; //Allume compteur
}
```

Algorithm 2 Approche-I

```
    void setupBoutons(){
    TRISE3=1;
        TRISC5=1;
        TRISD|=0b11010000;
}
    // lire ADC (potentiometer) - commande
    void setupAdcPot(){
        consigne=0;
        TRISA0=1; //AN0 entree POT1
        ANSEL=1; //AN0 analog, AN1-7 numerique
        ADCON1=0x10; //Fosc/8
        ADRESL=0;
        ADRESH=0;
        ADIF=0;
        ADIE=1;
        PEIE=1;
        ADCON0=1; //ADON, Channel 0 (AN0)
}
    //PWM - vitesse moteur
    void setupPWM(){
        TRISD2=0; //RD2 sortie PWM
        PR2=0xFF;
        TMR2ON=1;
        T2CKPS1=1;  //Prescaler mis a 16 pour le PWM donc Periode
      PWM=4.1ms
        T2CKPS0=1;
        CCP2M3=1; //PWM mode
        CCP2M2=1;
}
    void setupMoteur(){
        TRISD5=0; //MoteurON
        MoteurON=1;
}
    void setupCapteur(){
        TRISD1=1;
}
```

Algorithm 3 Approche-I

```
void interrupt gestion_IT() {
if(ADIF){
    consigne=ADRESH;
        GODONE=1; //debut ADC/CAN
        ADIF=0;
  }
    if(T0IF) {
        encoder=TMR1L;
        vitesse= encoder<<5; //Vitesse max = 8 flancs encoder en 32ms
     ->255 = Gain=255/8=32
        erreur=consigne-vitesse;
        commande=(erreur>>1)+(erreur>>2)+(erreur>>3);
        CCPR2H=0;
        CCPR2L=commande; // Modifie PWM
        TMR1ON=1;
        TMR1H=0; // Reinitialise Compteur
        TMR1L=0;
        T0IF=0;
  }
    if(TMR1IF) {
    //temps encoder trop grand - fixe a 255
        TMR1ON=0;
        TMR1H=0xFF;
        TMR1IF=0;
  }
}
    char i;
    void main(){
        setupMoteur();
        InitialisationAfficheur();
        setupT0();
        setupT1();
        setupBoutons();
        setupAdcPot();
        setupPWM();
        setupCapteur();
        GIE=1;
        GODONE=1; //debut ADC/CAN
        AfficherDecimal(0);
        while(1){ // Boucle infini
            i++;
            if(i>250){ // Affiche les infos selectionnes
                i=0;
                if(SwAffichConsigne==Appuye) {
                    AfficherDecimal(consigne);
             } else if(SwAffichErreur==Appuye) {
                    AfficherDecimal(erreur);
             } else if(SwAffichCommande==Appuye) {
                    AfficherDecimal(commande);
             } else if(SwAffichRetour==Appuye) {
                    AfficherDecimal(vitesse);
      }}}
}
```

Algorithm 4 Approche-II

```
/*********************************************************/
// µ-projet de commande par microcontrôleur
// ENS2M avril 2011
// Adrian BULNES
// David FUENMAYOR
/*********************************************************/

    #include <pic.h>
    #include "afficheur.h"
    #define SwAffichConsigne RE3
    #define SwAffichErreur RD4
    #define SwAffichCommande RD6
    #define SwAffichRetour RD7
    #define Capteur RD1
    #define Appuye 0
    #define MoteurON RD5
    #define Kp 1
    #define TestLed RA1
    static char counter;
    static char consigne;
    static char erreur;
    static char commande;
    static char retour;
    static char encoder;
    static char vitesse;
    //Timer 0 - interruptions tout les 4.1ms pour faire commande
    void setupT0() {
        //Fosc=4MHz, Fcy=1MHz, Tcy=1us
        //interruptions chaque 16*256us = 4.1ms@ prescaler 16
        T0CS=0; //Horloge Interne
        PSA=0;
        PS2=0; //Prescaler 16
        PS1=1; //Prescaler 16
        PS0=1; //Prescaler 16
        TMR0=0; //Initialization a zero
        T0IF=0; //Interrupt flag to zero
        T0IE=1; //Enable interrupt
}
    //Timer1 - Mode timer, compte nombre de flancs montants
    // TMR1H:TMRH1L = duration en microsecondes entre deux flancs du signal du
encoder
    // On se sert seulement de la partie superieure donc: 65ms > temps entre flancs >
0.256ms
    void setupT1() {
        encoder=0;
        vitesse=0;
        T1CKPS1=0; //Prescaler 1
        T1CKPS0=0; //Prescaler 1
        TMR1CS=0; //Mode Timer
        TMR1H=0; //Initialization a zero
        TMR1L=0; //Initialization a zero
```

Algorithm 5 Approche II

```
    TMR1IF=0;
        PEIE=1; //Enable peripheral interrupt
        TMR1IE=1;
        TMR1ON=1; //Allume compteur
}
    void setupBoutons(){
        TRISE3=1;
        TRISD|=0b11010000;
}
    // lire ADC (potentiometer) - commande
    void setupAdcPot(){
        consigne=0;
        TRISA0=1; //AN0 entree POT1
        ANSEL=1; //AN0 analog, AN1-7 numerique
        ADCON1=0x10; //Fosc/8
        ADRESL=0;
        ADRESH=0;
        ADIF=0;
        ADIE=1;
        PEIE=1;
        ADCON0=1; //ADON, Channel 0 (AN0)
}
    //PWM - vitesse moteur
    void setupPWM(){
        TRISD2=0; //RD2 sortie PWM
        PR2=0xFF;
        TMR2ON=1;
        T2CKPS1=1;
        T2CKPS0=1;  //Prescaler mis a 16 pour le PWM donc Periode-
      PWM=4.1ms
        CCP2M3=1; //PWM mode
        CCP2M2=1;
}
    void setupMoteur(){
        TRISD5=0; //MoteurON
        MoteurON=1;
}
    void setupCapteur(){
        TRISD1=1;
}
    void interrupt gestion_IT() {
        if(ADIF){
            consigne=ADRESH;
            if(consigne>250) consigne=255;
            GODONE=1; //debut ADC/CAN
            ADIF=0;
      }
```

Algorithm 6 Approche II

```
        else if(T0IF) {
            erreur=consigne-(255-encoder);
            commande=(erreur>>1)+(erreur>>2)+(erreur>>3);
            CCPR2H=0;
            CCPR2L=commande;
            T0IF=0;
    } else if(TMR1IF) {
            //temps encoder trop grand - fixe a 255
            TMR1ON=0;
            TMR1H=0xFF;
            TMR1IF=0;
            }
    }
        char i;
        void main(){
            setupMoteur();
            InitialisationAfficheur();
            setupT0();
            setupT1();
            setupBoutons();
            setupAdcPot();
            setupPWM();
            setupCapteur();
            TRISA1=0;
            GIE=1;
            //debut ADC/CAN
            GODONE=1;
            AfficherDecimal(0);
            while(1){ //infinite loop
                // Software debouncing code
                if (counter < 255){ //previent que le compteur reviens a zero
                    counter++;
             }
                //capteur - falling edge
                if(Capteur==1){
                    counter=0;
                    TestLed=0;
             }
                if (counter == 20) { //Apres stabilization et une seule fois
                //lire temp encoder (capteur)
                    TestLed=1;
                    encoder=TMR1H;
                    TMR1ON=1;
                    TMR1H=0;
                    TMR1L=0;
                    counter++;
             }
```

Algorithm 7 Approche II

```
        i++;
            if(i>200){
                i=0;
                if(SwAffichConsigne==Appuye){
                    AfficherDecimal(consigne);
             } else if(SwAffichErreur==Appuye){
                    AfficherDecimal(erreur);
             } else if(SwAffichCommande==Appuye){
                    AfficherDecimal(commande);
             } else if(SwAffichRetour==Appuye){
                    AfficherDecimal(encoder);
             }
          }
    }
        }
```